AWS re:Invent

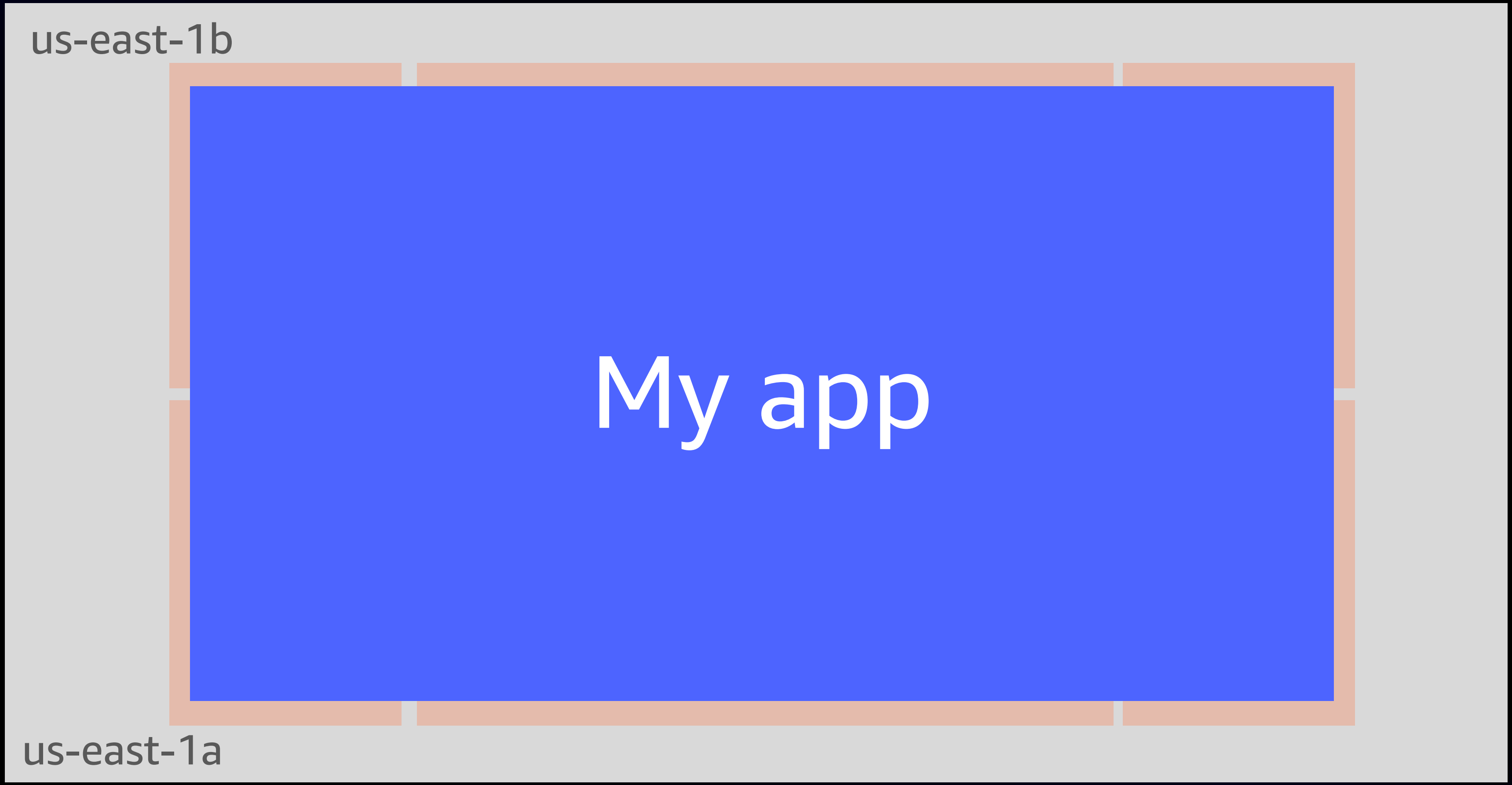
NOV. 29 – DEC. 3, 2021 | LAS VEGAS, NV

API308

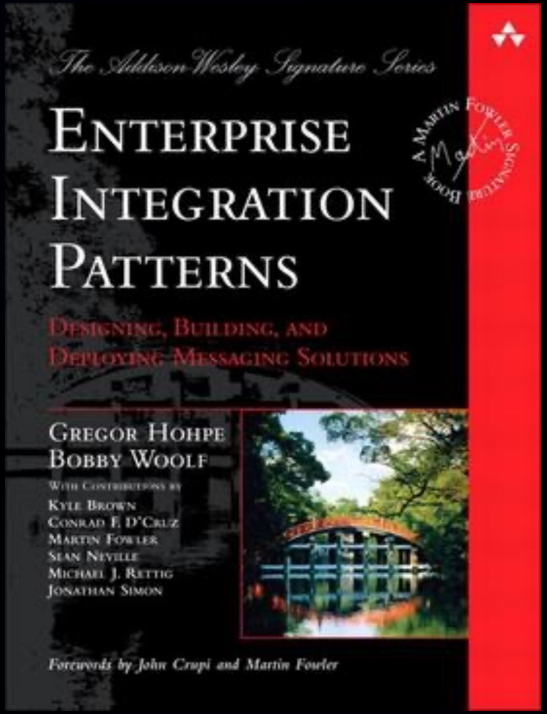
Building modern cloud applications? Think integration

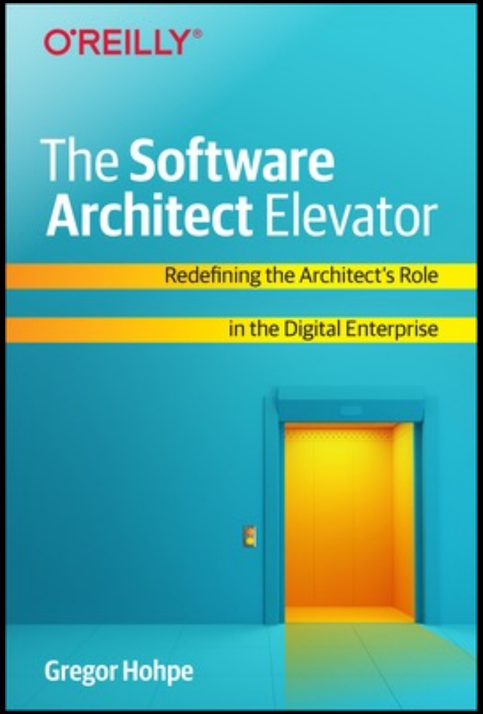
Gregor Hohpe
Enterprise Strategist
Amazon Web Services

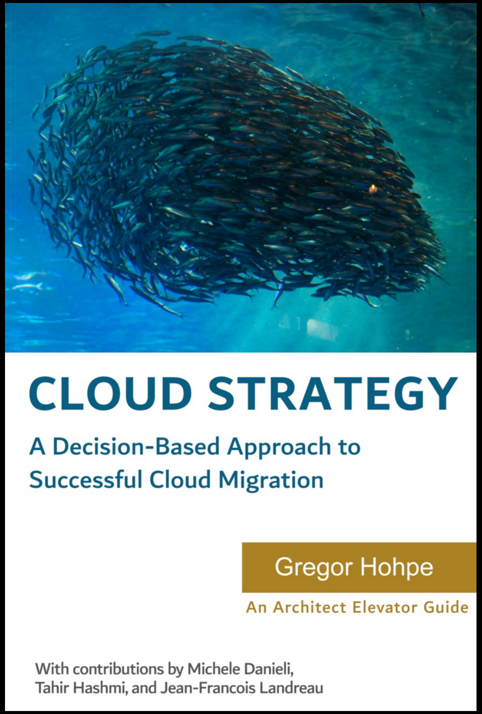
My modern cloud application



Gregor Hohpe – Enterprise Strategist


The Addison Wesley Signature Series
Enterprise Integration Patterns
Designing, Building, and Deploying Messaging Solutions
Gregor Hohpe
Bobby Woolf
With contributions by
Kyle Brown
Conrad F. D'Cruz
Martin Fowler
Sean Neville
Michael J. Rettig
Jonathan Simon
Forewords by John Crupi and Martin Fowler
A Martin Fowler Signature Book


O'REILLY
The Software Architect Elevator
Redefining the Architect's Role
in the Digital Enterprise
Gregor Hohpe

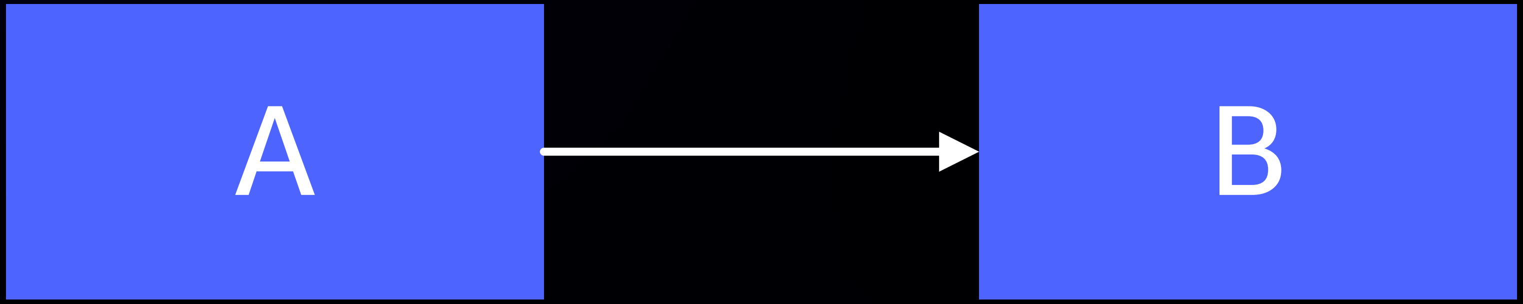

CLOUD STRATEGY
A Decision-Based Approach to Successful Cloud Migration
Gregor Hohpe
An Architect Elevator Guide
With contributions by Michele Danieli, Tahir Hashmi, and Jean-Francois Landreau

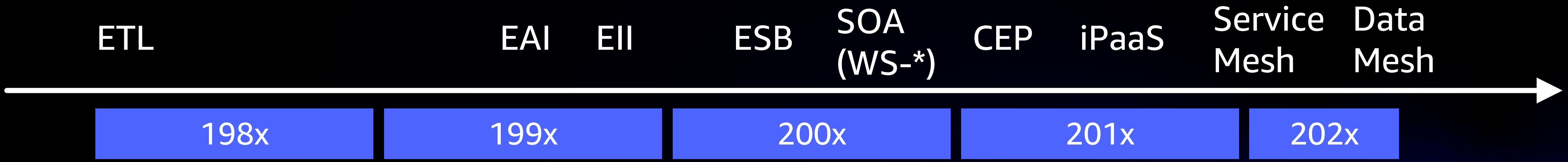
As an AWS Enterprise Strategist, Gregor helps enterprise leaders rethink their IT strategy to get the most out of their cloud journey.

@ghohpe
ArchitectElevator.com
www.linkedin.com/in/ghohpe/

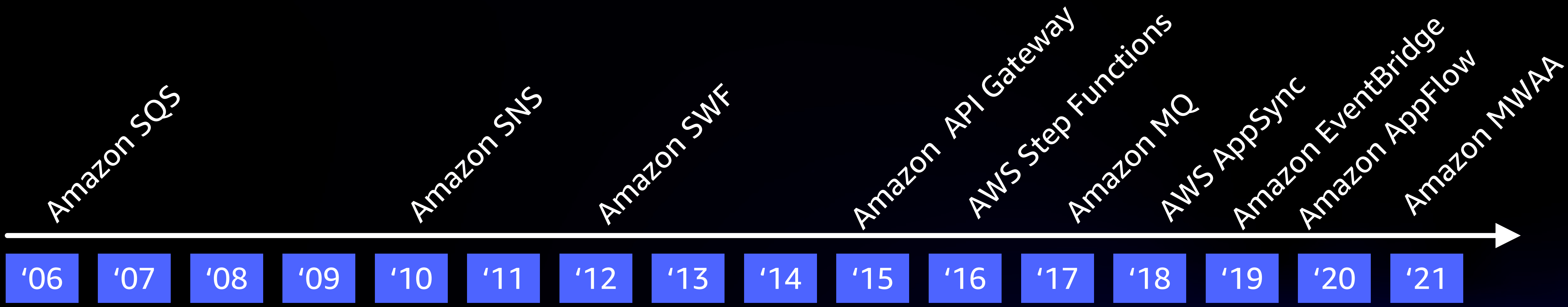
Application integration in the cloud

Connected Systems – Concept & reality





History of AWS Integration Services



Cataloging integration approaches

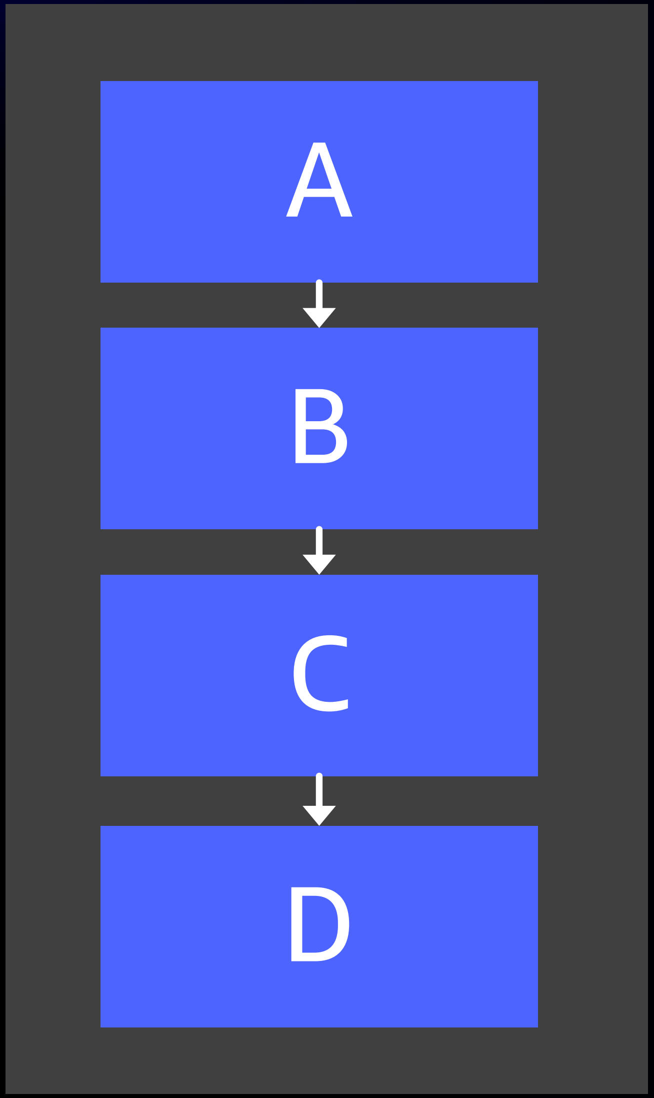
Approach	Level of Control	Delivery Lifecycle	Team	Example (indicative)
Migration	Low	One-time	One-off	Amazon AppFlow Amazon SWF
Data synchronization / traditional integration	Low	Slow	Separate	Amazon AppFlow
Enterprise service bus	Some	Faster (slower than endpoints)	Likely separate	Amazon MQ, Amazon SQS, Amazon API Gateway
Modern cloud apps serverless EDA	High	Same pace	Embedded	Amazon EventBridge, AWS Step Functions

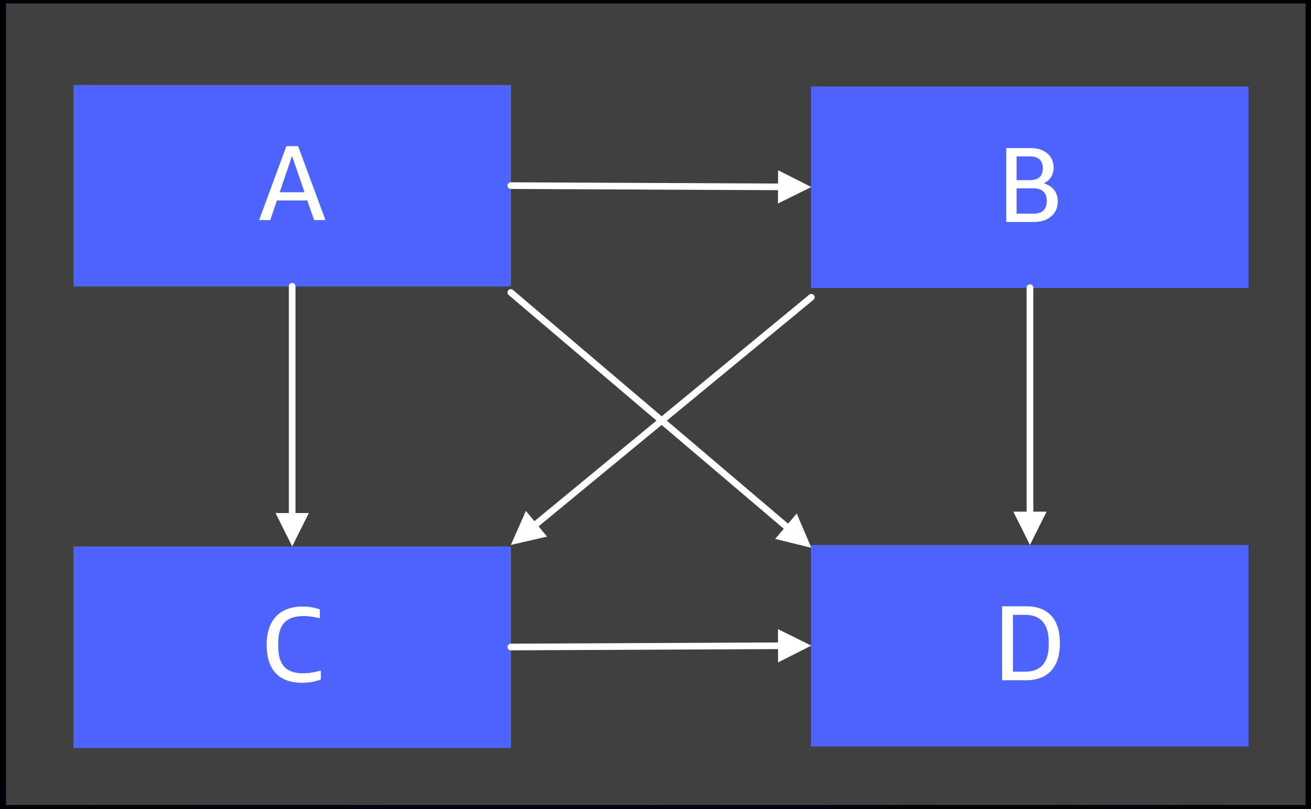
“In modern cloud applications, integration isn’t an afterthought.

It’s an integral part of the application architecture and the software delivery lifecycle.”

Of boxes and lines

Two system designs



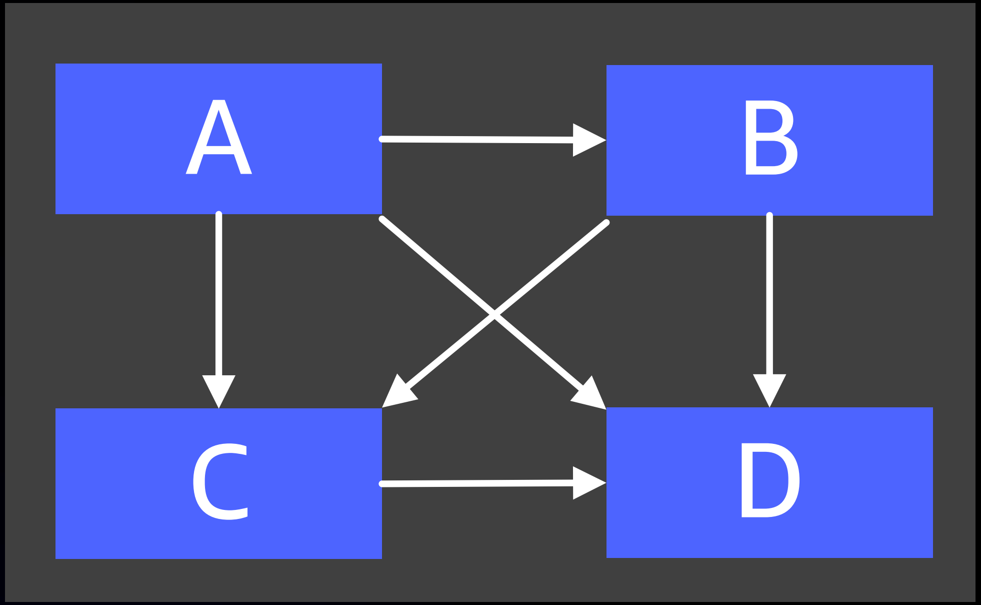


“How your components are interconnected defines your system’s essential properties.”

Software Systems Architecture

"The fundamental structures of a software system and the discipline of creating such structures and systems.
Each structure comprises software elements, relations among them, and properties of both elements and relations."

Documenting Software Architectures
Clements, Bass, Garlan, et al.

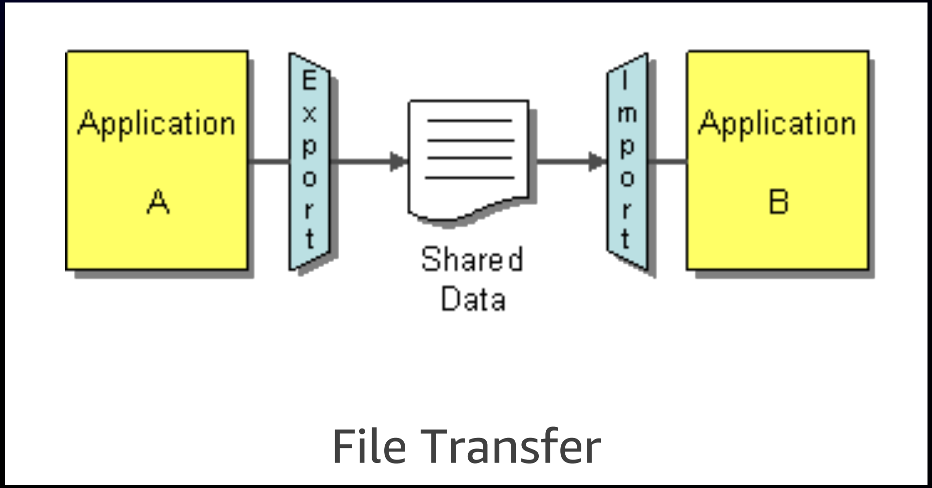


“Great architects are like great chefs.
It’s not just about selecting ingredients; it’s how you put them together.”

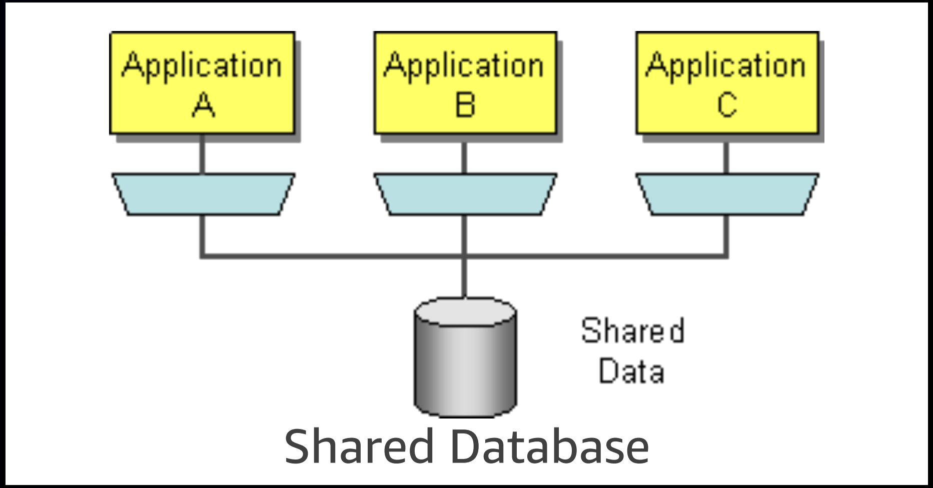
Gregor

Integration Architecture: Considerations

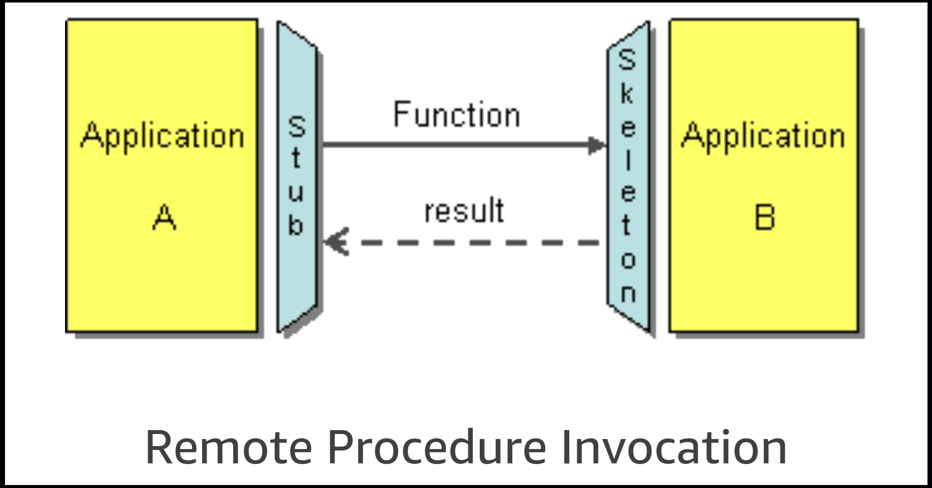
Integration approaches



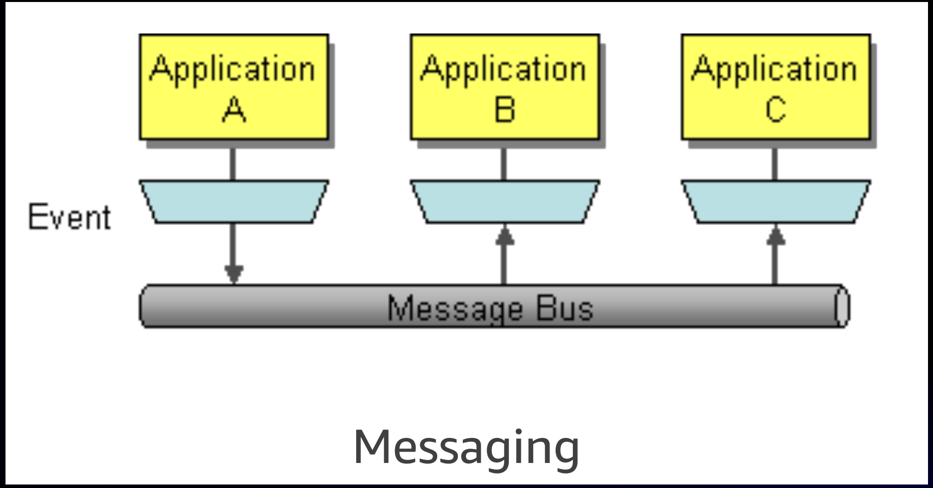
File Transfer



Shared Database



Remote Procedure Invocation



Messaging

Coupling

Abstraction

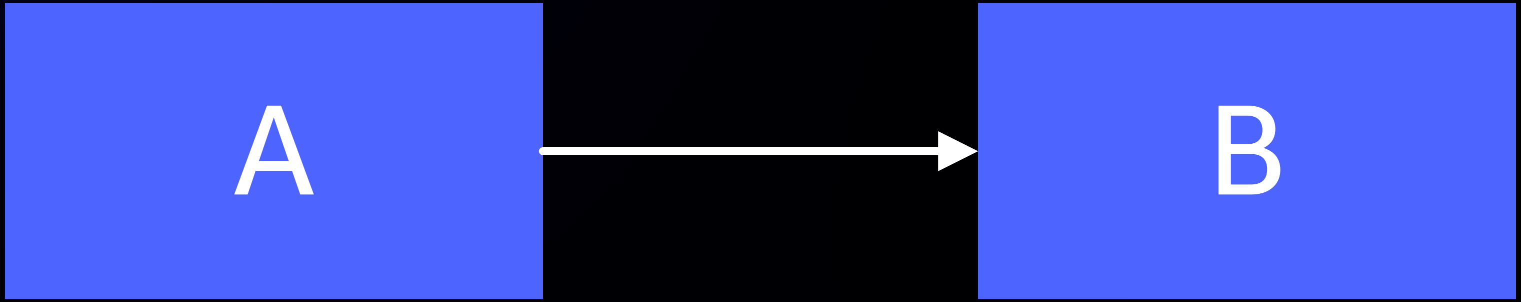
Asynchrony

Timeliness

Complexity

Source: Enterprise Integration Patterns

Coupling – Integration's magic word



Coupling is a measure of independent variability between connected systems.

Decoupling has a cost, both at design and run-time.

Coupling isn't binary.

Coupling isn't one-dimensional.

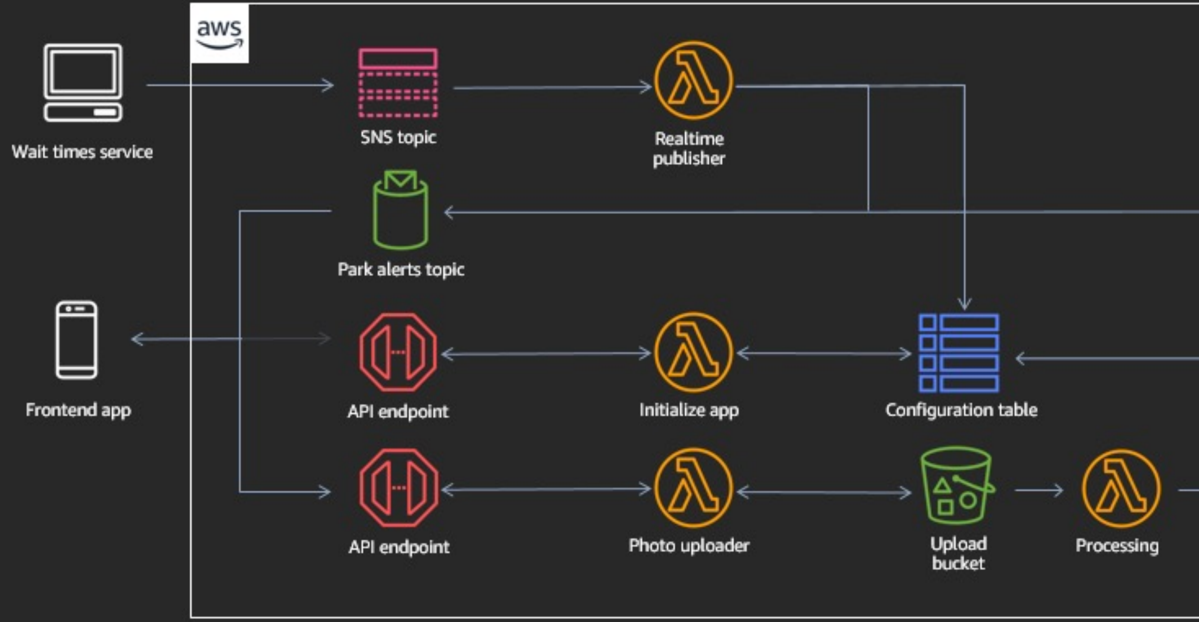
The many facets of coupling

• Technology dependency:	Java vs. C++
• Location dependency:	IP addresses, DNS
• Data format dependency:	Binary, XML, JSON, ProtoBuf, Avro
• Data type dependency:	int16, int32, string, UTF-8, null, empty
• Semantic dependency:	Name, Middlename, ZIP
• Temporal dependency:	sync, async
• Interaction style dependency:	messaging, RPC, query-style (GraphQL)
• Conversation dependency:	pagination, caching, retries

"The appropriate level of coupling depends on the level of control you have over the endpoints."

Modern Cloud Applications

Small pieces, loosely joined



Serverless is much more than compute

COMPUTE

AWS Lambda

AWS Fargate

DATA STORES

Amazon S3

Amazon Aurora Serverless

Amazon DynamoDB

INTEGRATION

Amazon EventBridge

Amazon API Gateway

Amazon SQS

Amazon SNS

AWS Step Functions

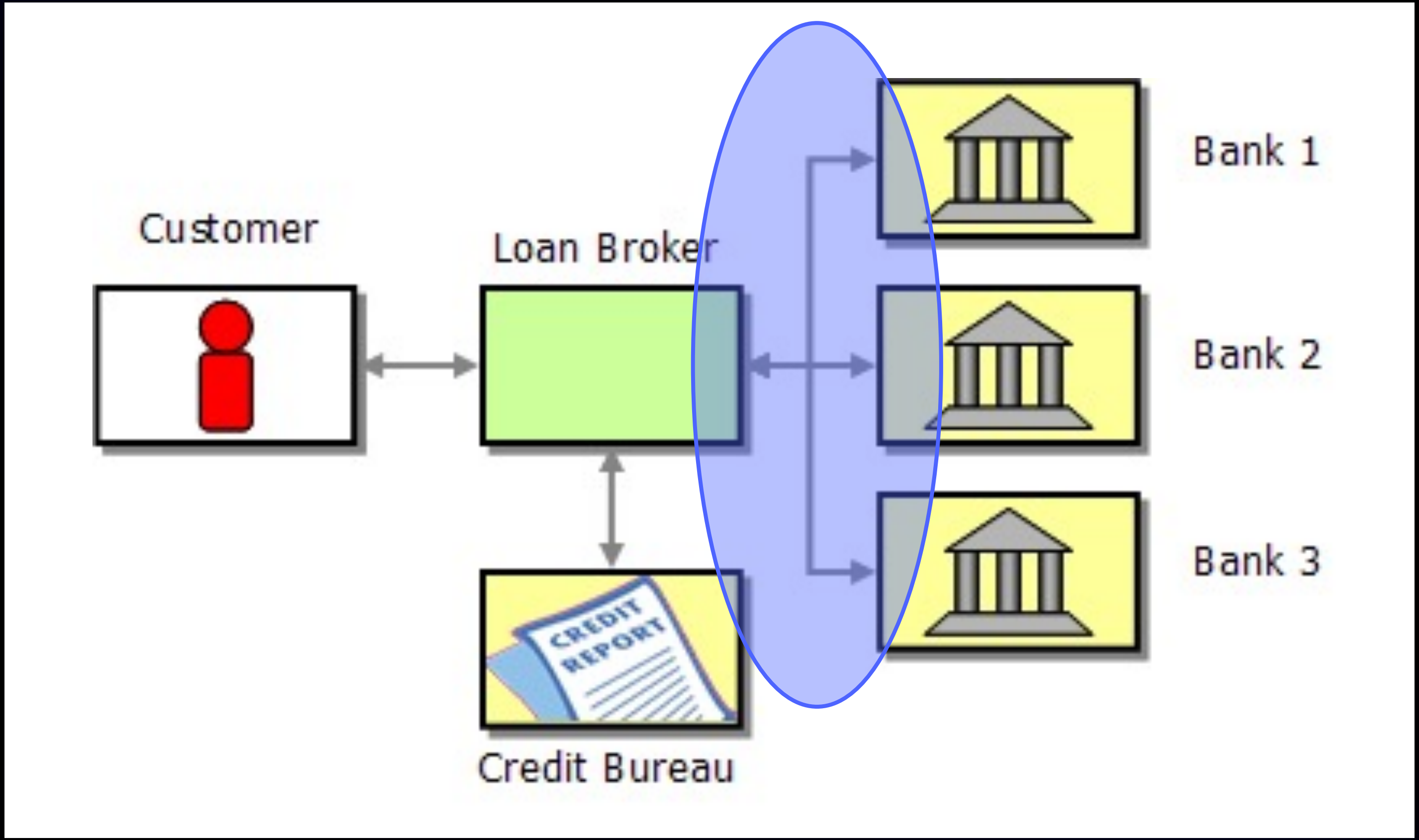
AWS AppSync

“Serverless is about much more than application run-times.

Modern serverless architectures are inherently integrated.”

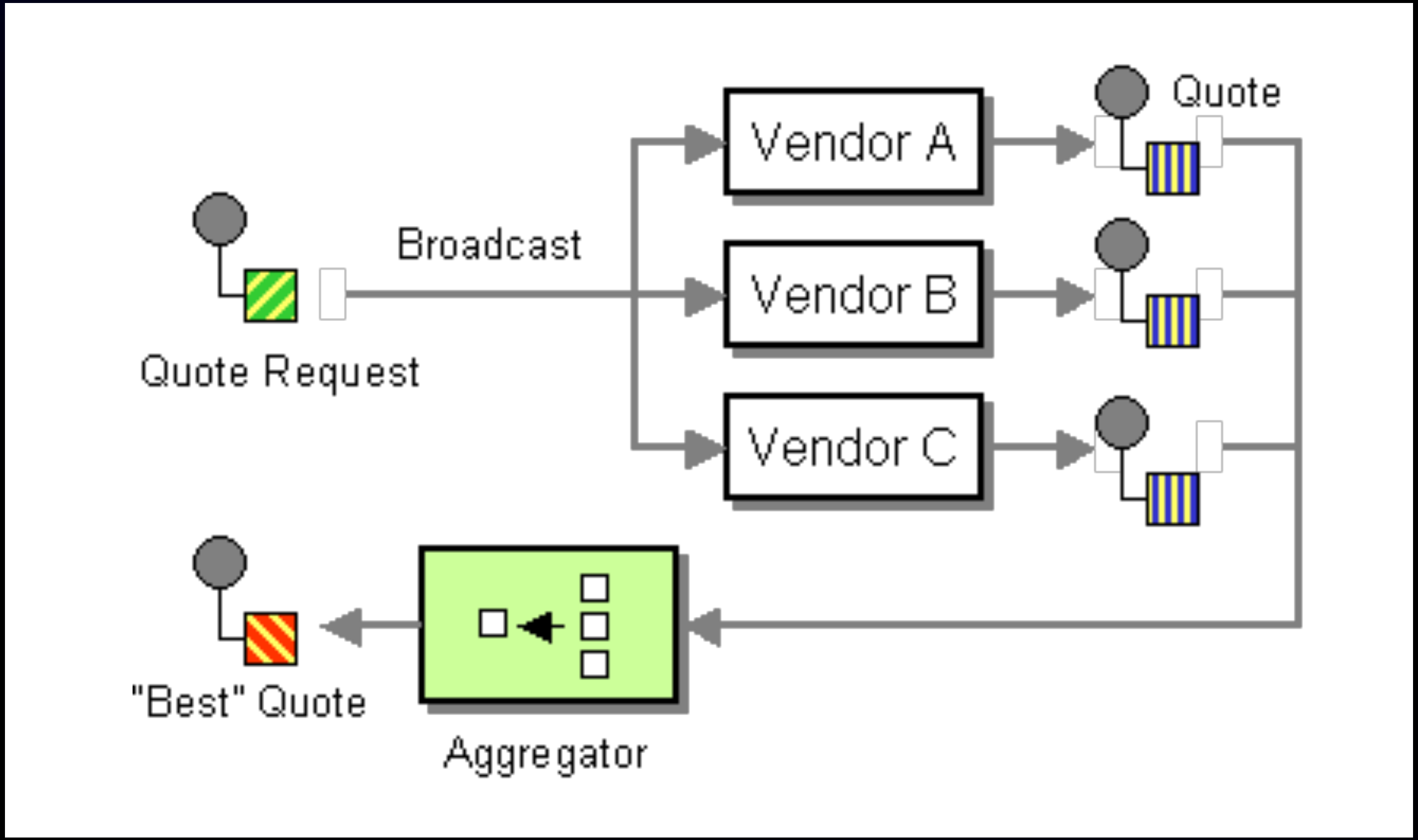
A modern cloud application: The Loan Broker

A simple distributed application



Source: EnterpriseIntegrationPatterns.com

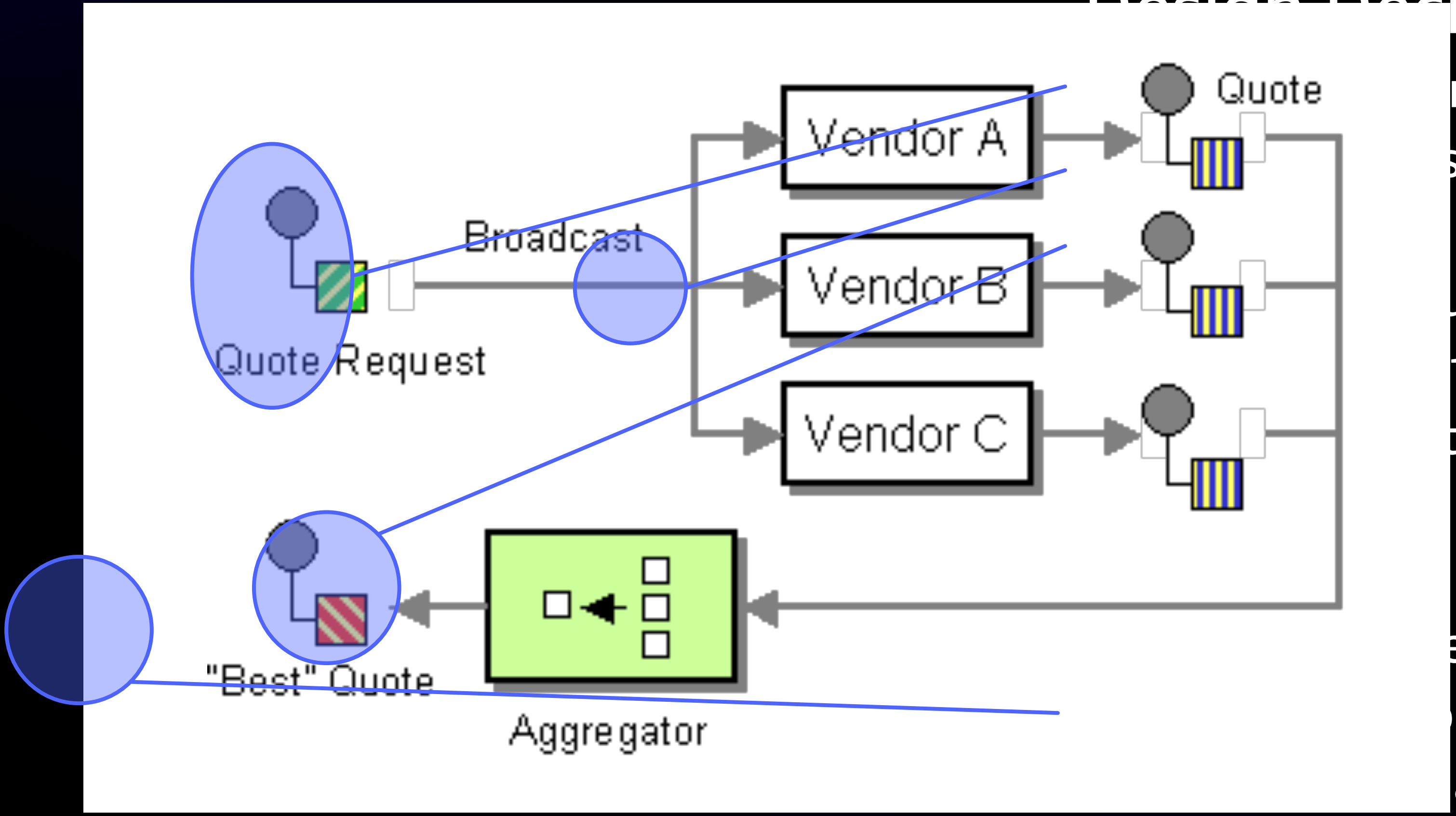
The central pattern: Scatter-Gather



"How do you maintain the overall message flow when a message needs to be sent to multiple recipients, each of which may send a reply?"

https://www.enterpriseintegrationpatterns.com/patterns/messaging/BroadcastAggregate.html

The central pattern: Scatter-Gather



Design Decisions:

mine set of recipients

s required to respond?

plete aggregation:

umber ("wait for all")

t ("time-out")

number of responses

(1 is sufficient)

of favorable response

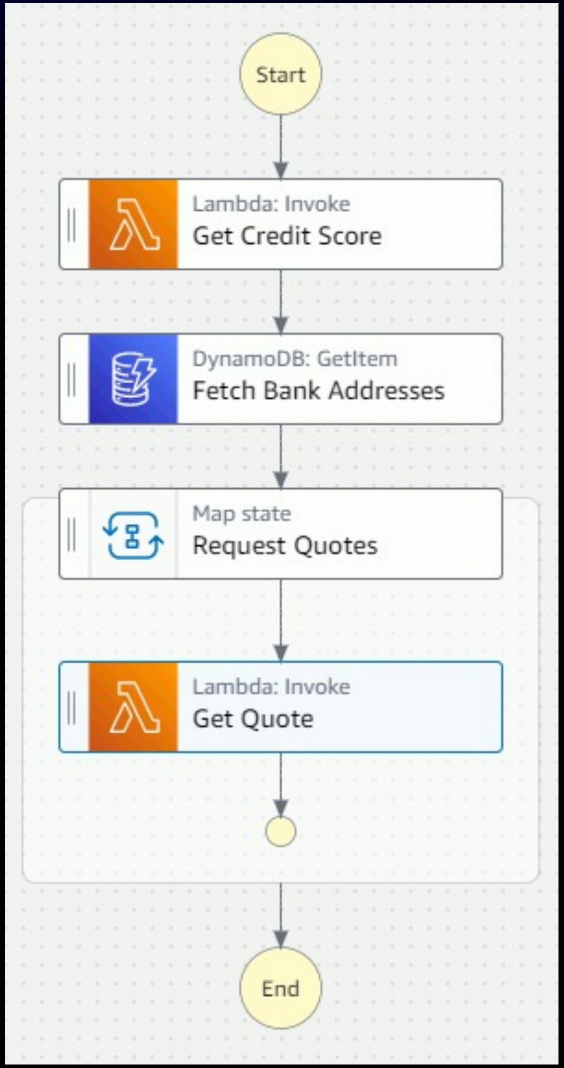
event ("gavel drops")

ine responses

ate

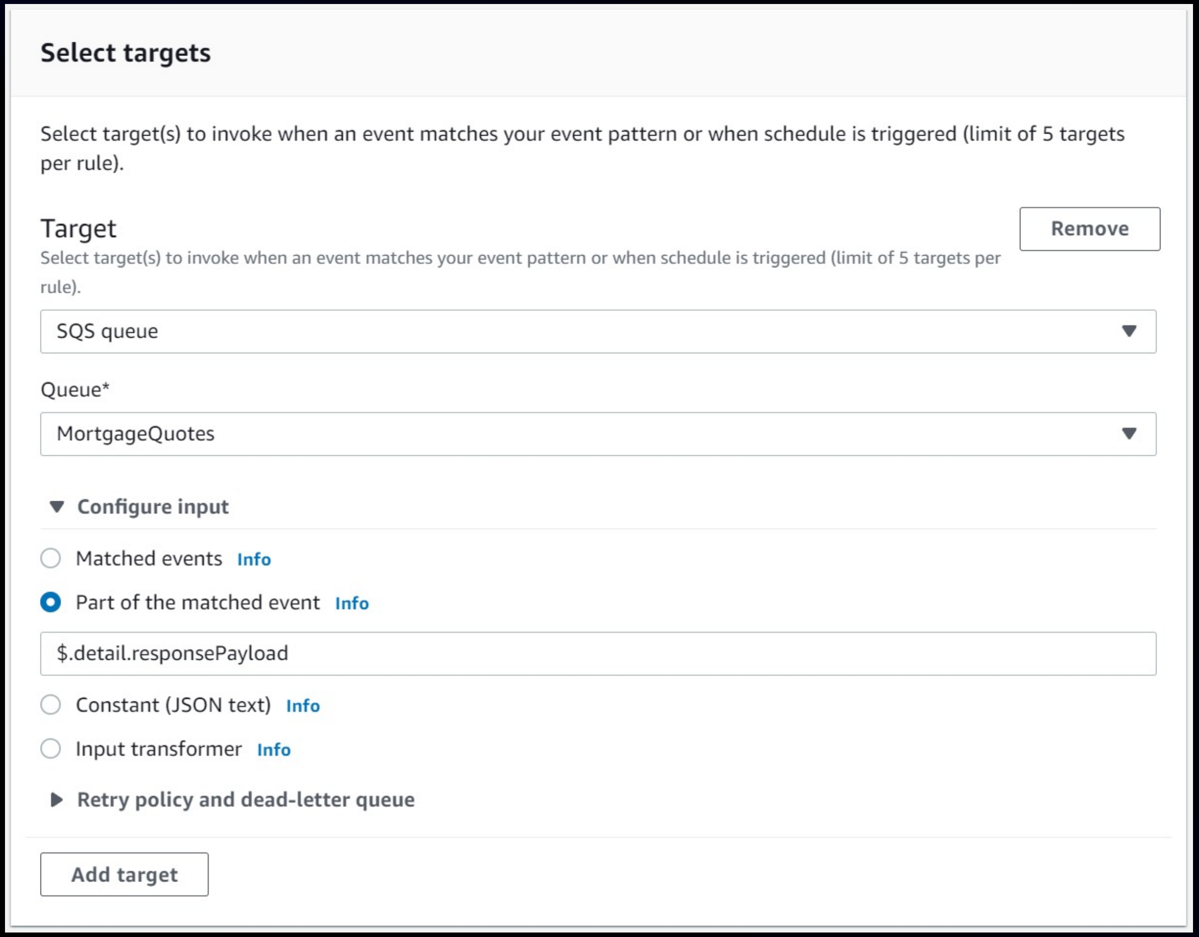
- Select best answers
- Combine answers (sum, avg)

Scatter-Gather: Step Functions Map State



- Fetch Lambda function names from DynamoDB table
- Iterate over the list, invoking Lambda functions (synchronously but concurrently - `MaxConcurrency`)
- Filter results to just bank ID and rate
- Concatenate results as they are returned (Map State does that for us)

Scatter-(Gather): EventBridge Targets



Select targets

Select target(s) to invoke when an event matches your event pattern or when schedule is triggered (limit of 5 targets per rule).

Target

Select target(s) to invoke when an event matches your event pattern or when schedule is triggered (limit of 5 targets per rule).

SQS queue

Queue*

MortgageQuotes

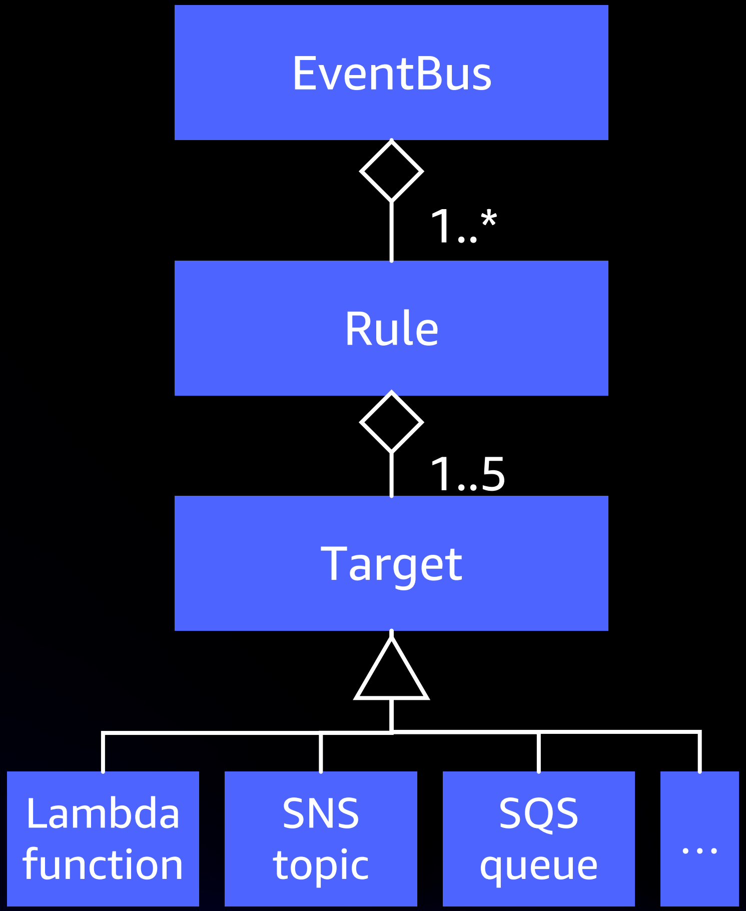
▼ Configure input

- Matched events
- [✓ Part of the matched event]

$.detail.responsePayload

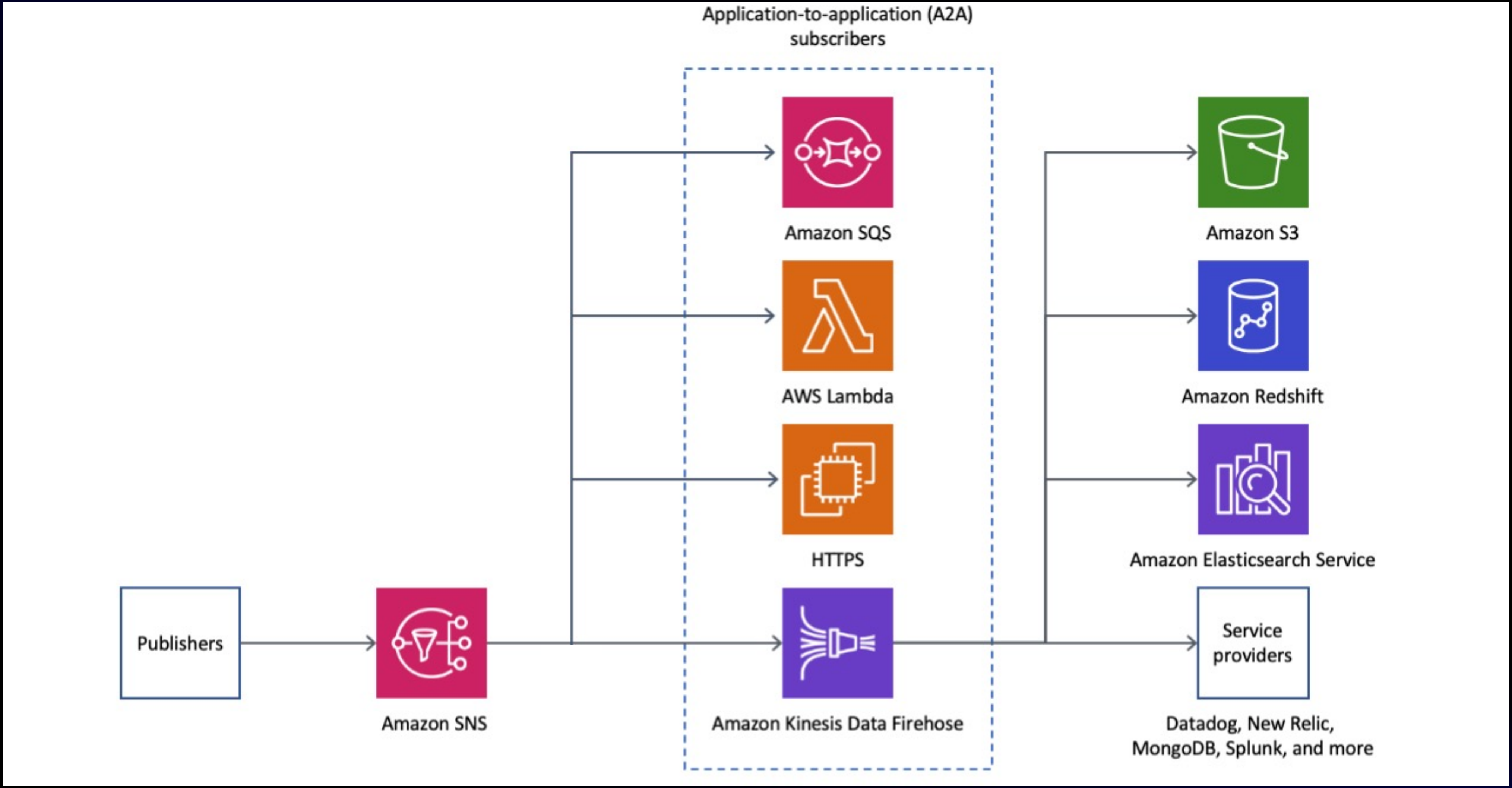
- Constant (JSON text)
- Input transformer

▶ Retry policy and dead-letter queue



Yup, UML!

Scatter-(Gather): Simple Notification Service


Application-to-application (A2A) subscribers
Amazon SQS
AWS Lambda
HTTPS
Amazon Kinesis Data Firehose
Publishers
Amazon SNS
Amazon S3
Amazon Redshift
Amazon Elasticsearch Service
Service providers
Datadog, New Relic, MongoDB, Splunk, and more

Comparing implementations (simplified)

	Fan-out	Interaction Model	Subscription Method	Entity	Visibility	Coupling
Step Functions Map/DynamoDB	Medium (dozens)	Sync / parallel	DynamoDB: UpdateItem	Sender	Poor	Loose
EventBridge	Small (5 per rule)	Sync	EventBridge: PutTargets	Broker	Good	Tight (?)
SNS	Large (12.5 mio per topic)	Async / push	SNS:Subscribe	Channel	Good	Loose

Subscription via: API / CLI / CFN / CDK / Console

“Fine-grained serverless applications make the solution’s intent explicit.”

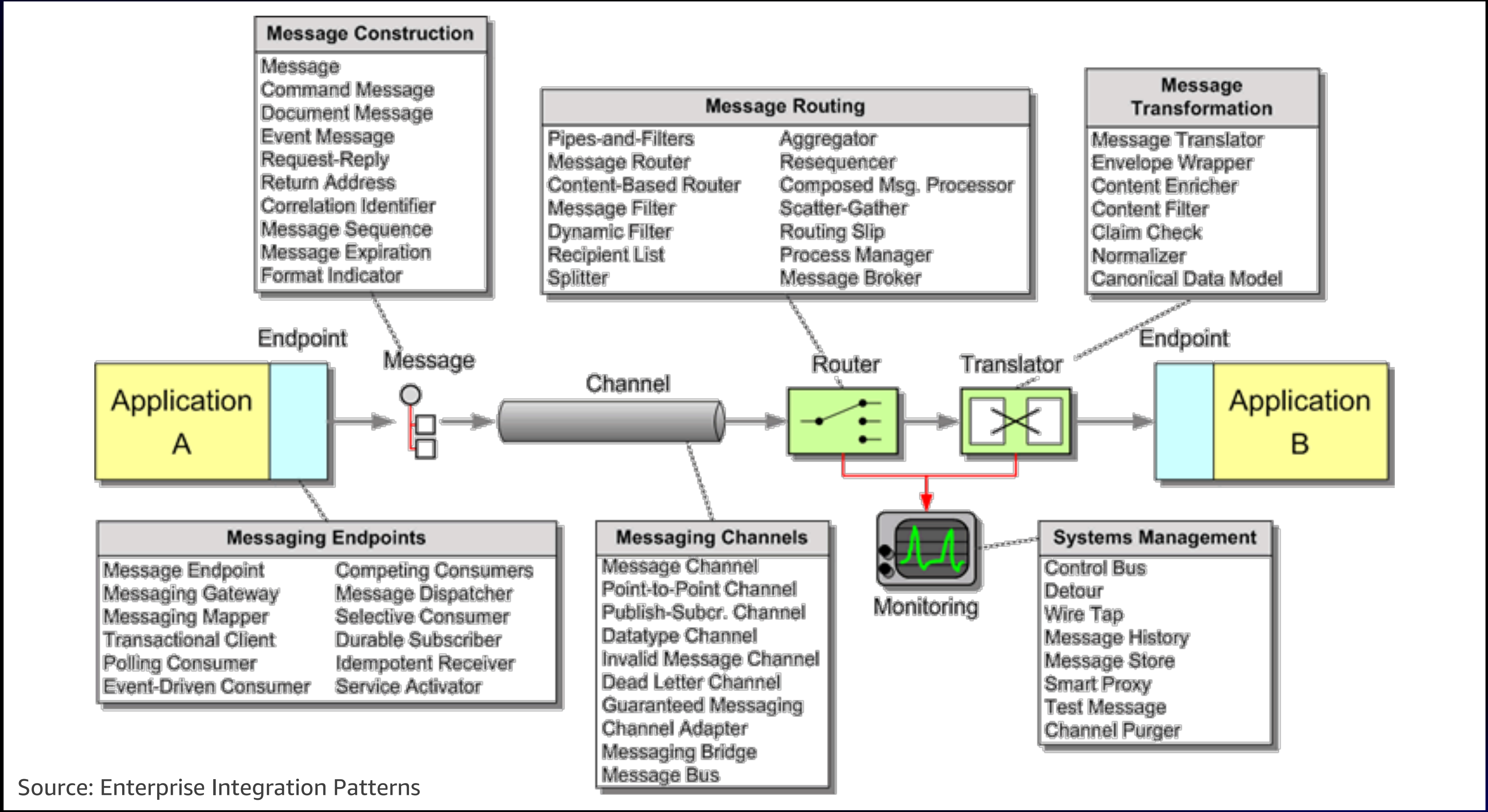
The Power of Patterns

Design Patterns

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
Addison-Wesley Professional Computing Series
Pattern-Oriented Software Architecture
A System of Patterns
Volume 1
Wiley
Patterns of Enterprise Application Architecture
Martin Fowler
Enterprise Integration Patterns
Designing, Building, and Deploying Messaging Solutions
Gregor Hohpe
Bobby Woolf
1994
1996
2002
2003

- ☑ Known solution to a recurring problem within a given context
- ☑ Bite-size, technology-independent design wisdom
- ☑ Express intent, the “why”, not just the “how”
- ☑ Shared vocabulary to express design choices and trade-offs

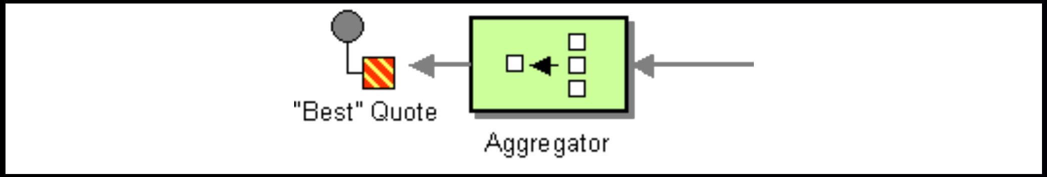
A Pattern *Language* for Integration



Source: Enterprise Integration Patterns

Integration Patterns in AWS Integration

Integration patterns to express your solution

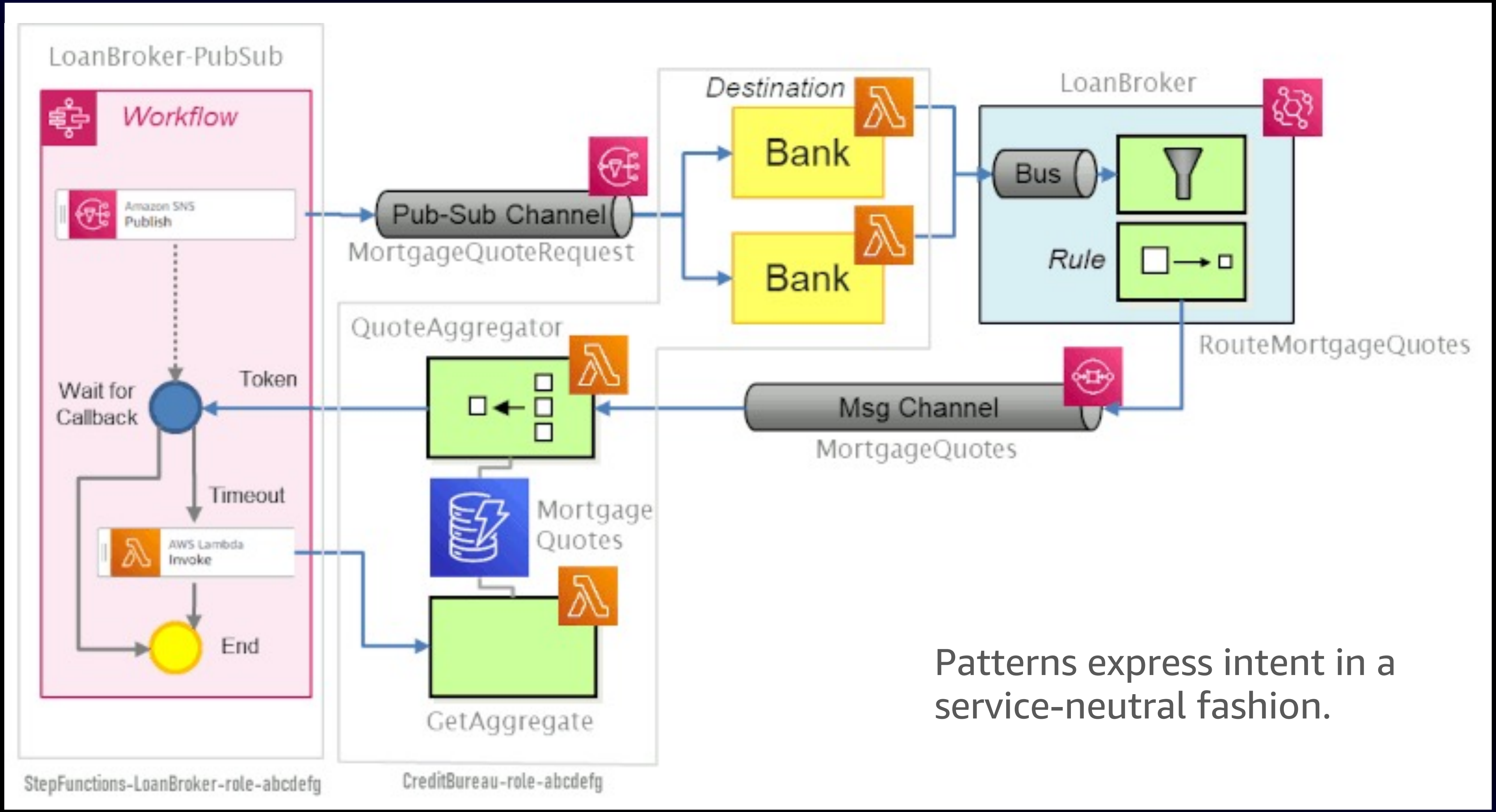


- Better express design decisions and trade-offs
- Hide implementation details
- Create visually appealing diagrams

Integration Patterns built into AWS services

- Straightforward mapping to AWS service
- Easier learning curve
- Better composability of services

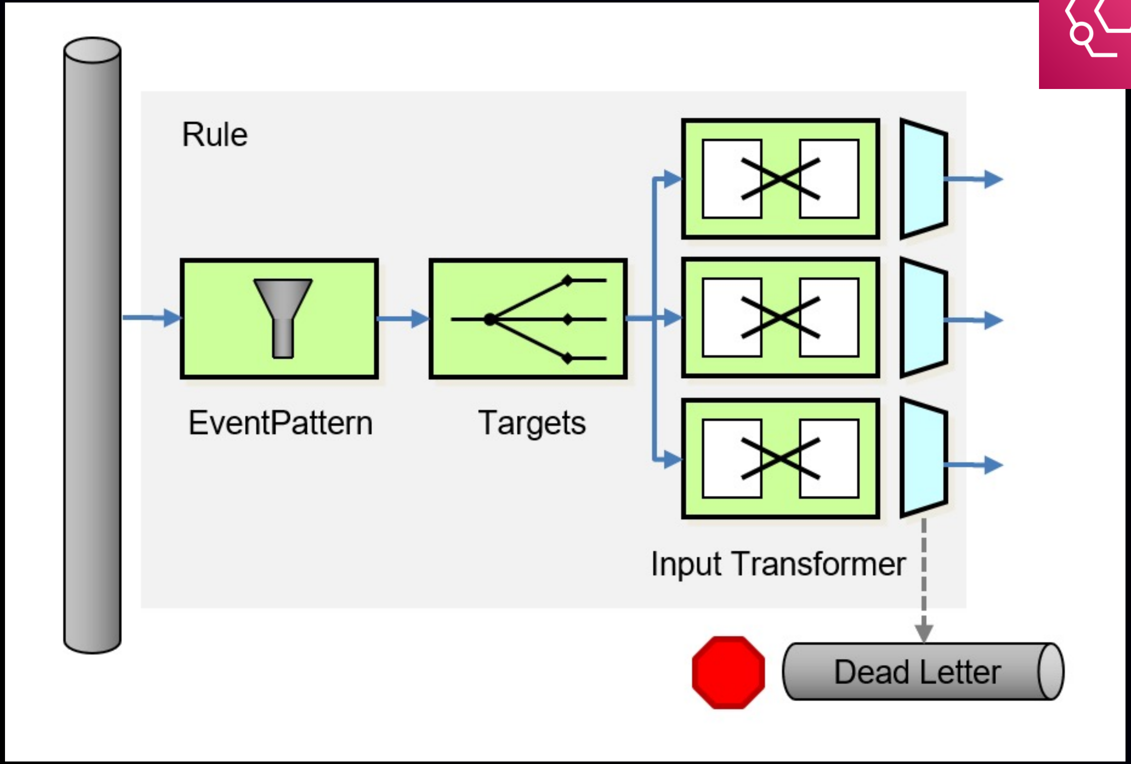
A cloud-native serverless implementation



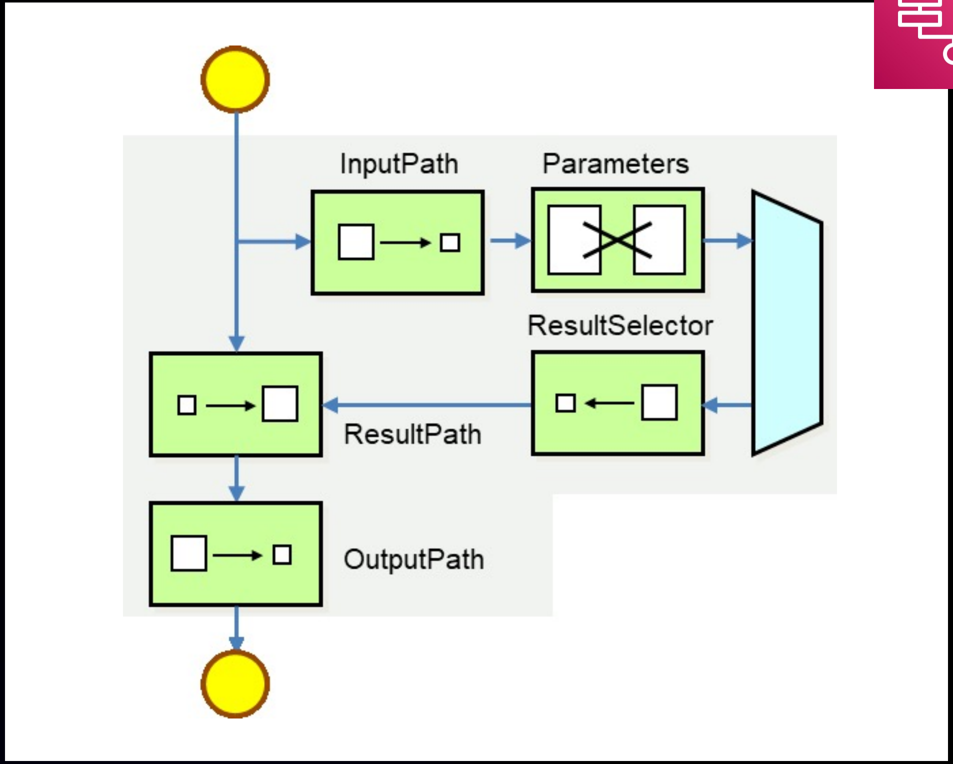
Patterns express intent in a service-neutral fashion.

- Scatter-Gather
- Pub-Sub Channel
- Msg Channel
- Message Filter
- Content Filter
- Aggregator
- Dead-Letter Queue

Integration Patterns in AWS Serverless

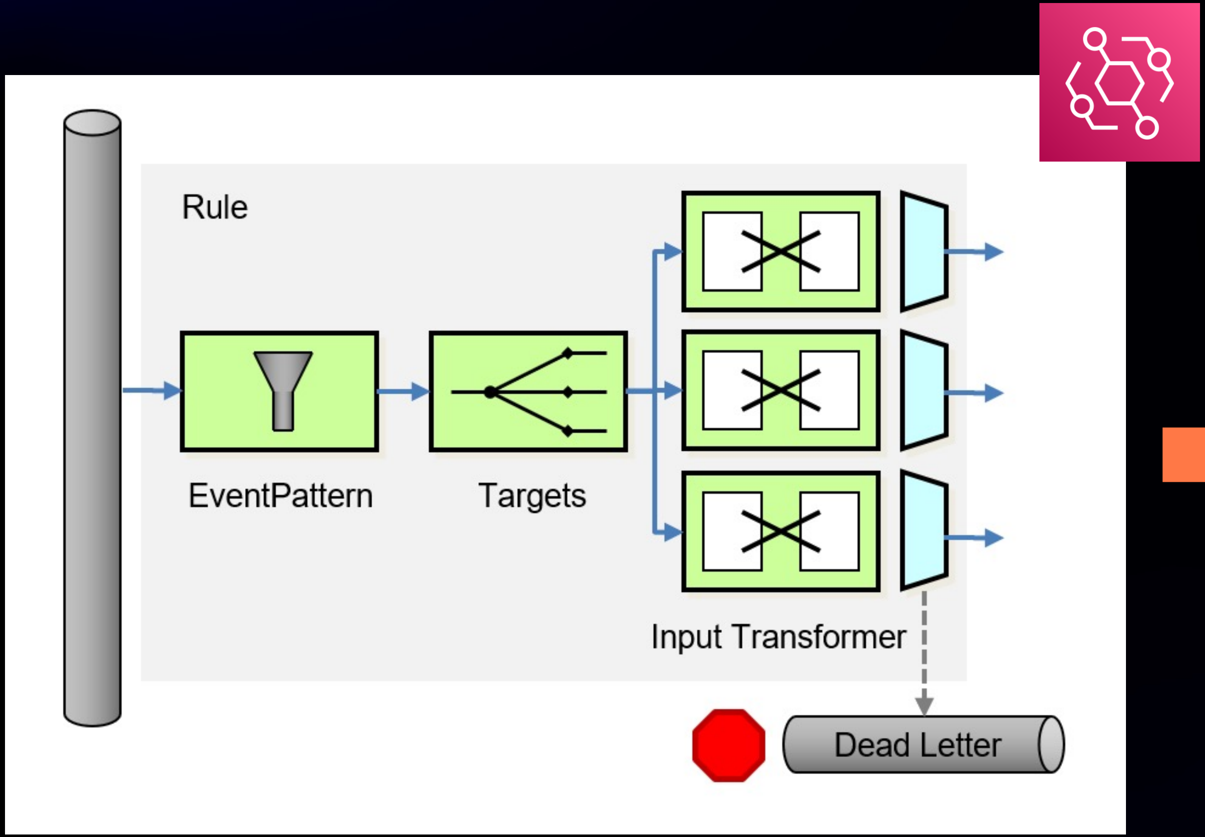


Amazon EventBridge

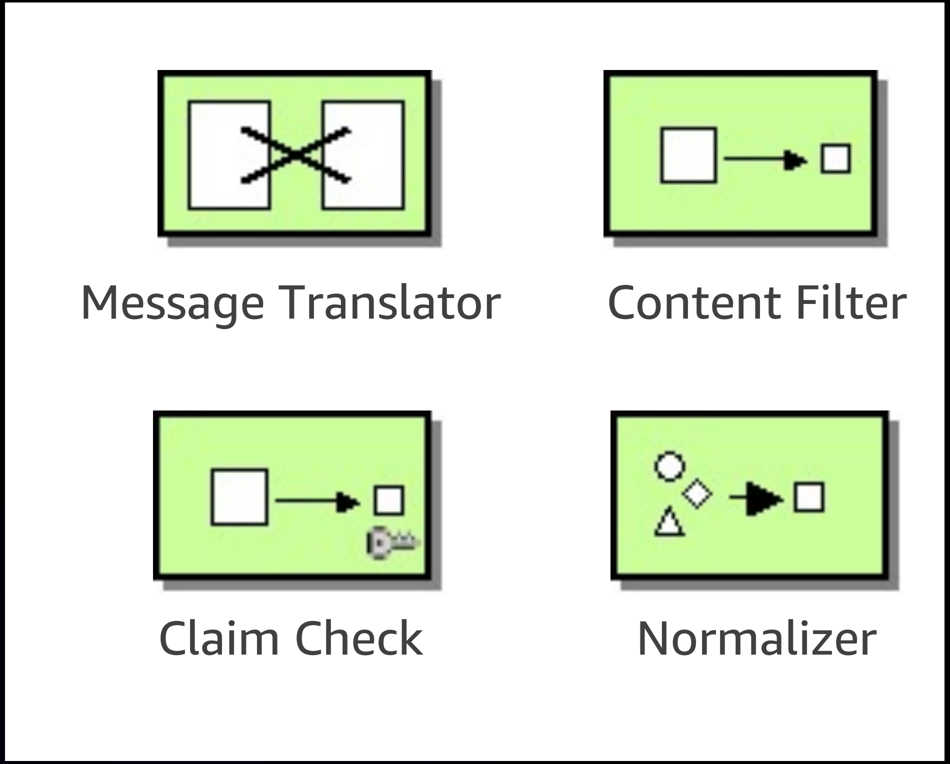


AWS Step Functions

Patterns express intent and nuances



Amazon EventBridge



Messaging Patterns

Source: Enterprise Integration Patterns

(Not just) Infrastructure as (actual) code

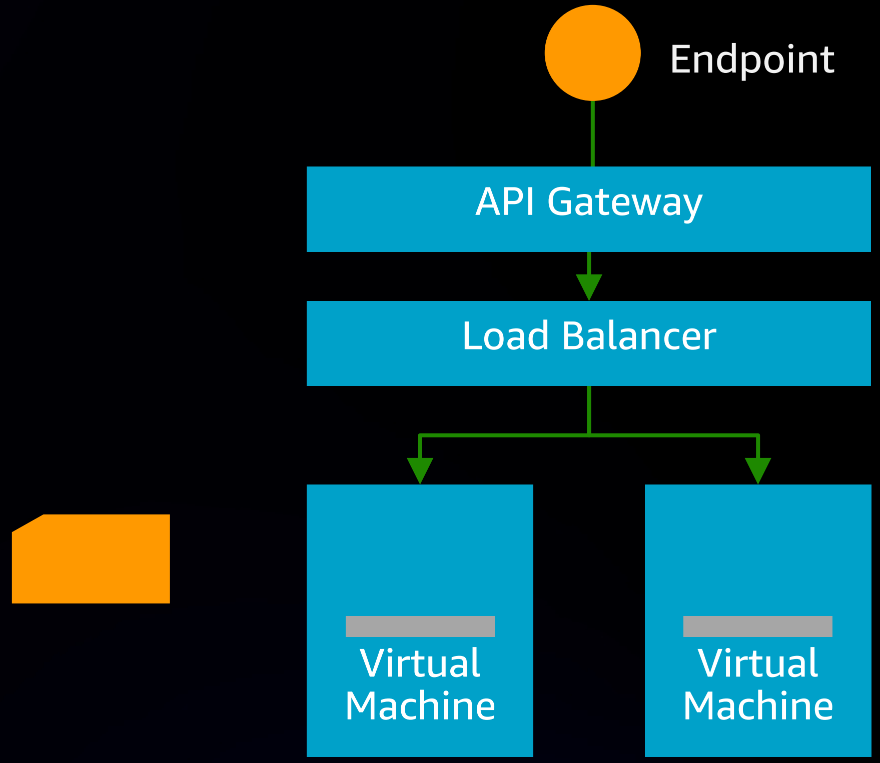
An automation stack

Config

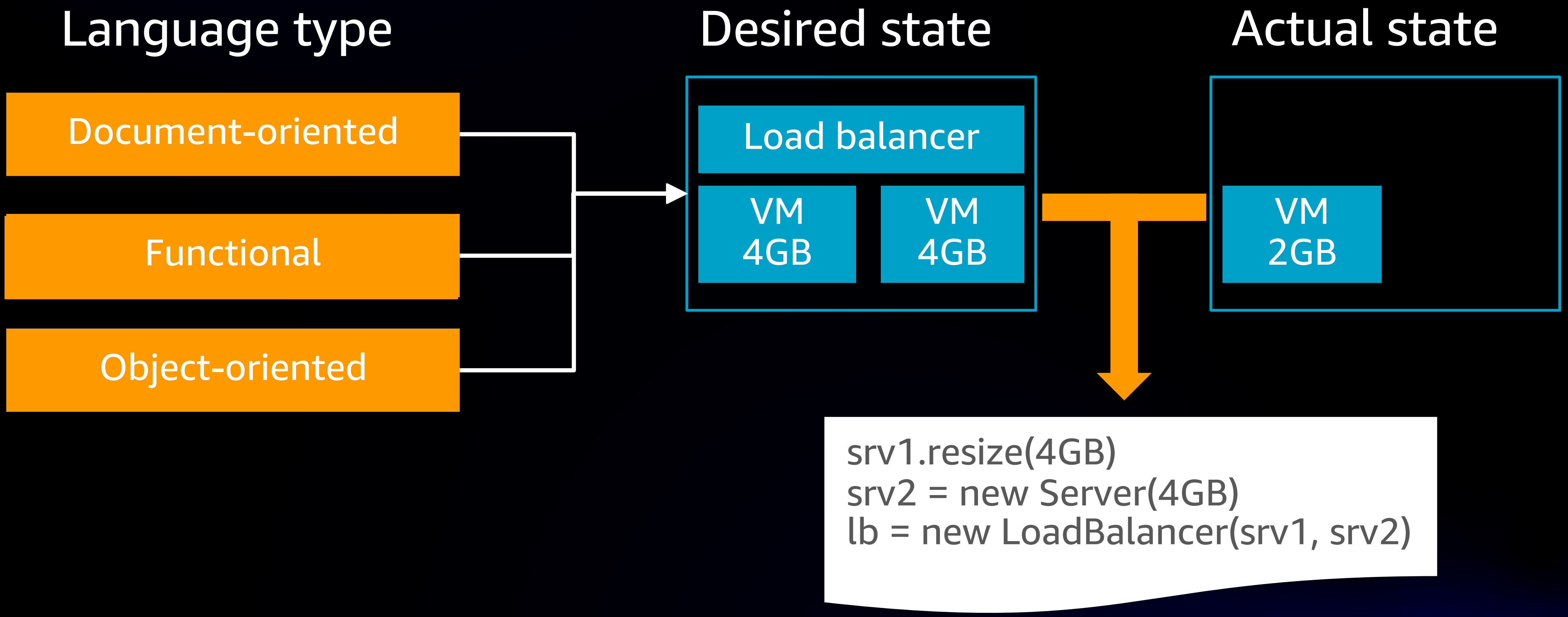
Compose

Deploy

Provision



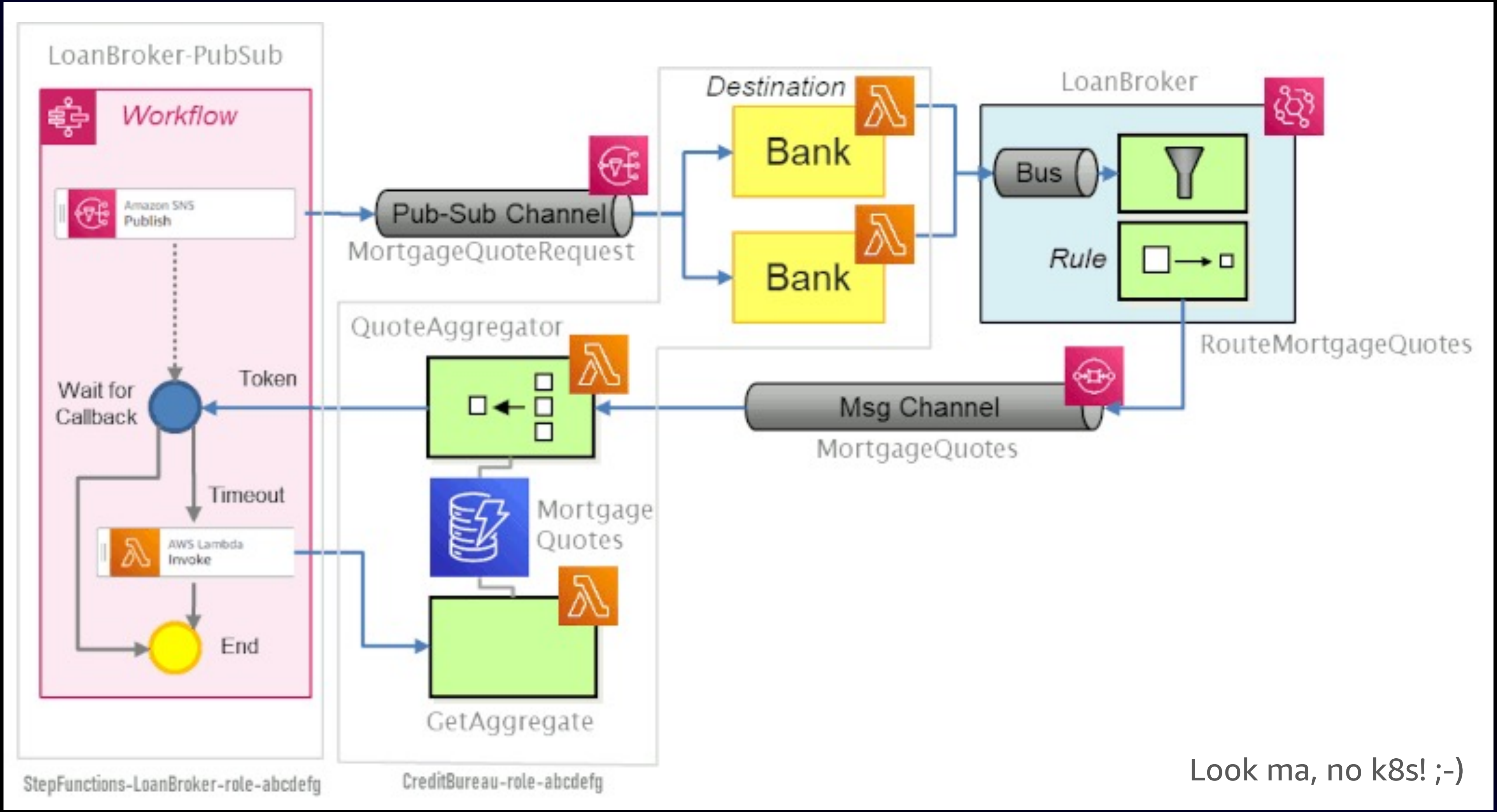
Declarative provisioning ≠ Declarative language



```
srv1.resize(4GB)
srv2 = new Server(4GB)
lb = new LoadBalancer(srv1, srv2)
```

Serverless + integration + automation = AWSome!

A cloud-native serverless implementation



Serverless Composition with CDK

Deployment

Config

```
const bankRecipientPawnshop = this.createBank(
  'BankRecipientPawnshop', { BANK_ID: 'PawnShop', BASE_RATE: '5',
  MAX_LOAN_AMOUNT: '500000', MIN_CREDIT_SCORE: '400‘ }, mortgageQuotesBus);
```

Composition

```
private createBank(name: string, env: BankConfig, eventBus: EventBus) {
  return new lambda.Function(this, name, {
    runtime: lambda.Runtime.NODEJS_14_X,
    code: lambda.Code.fromAsset('bank'), handler: 'app.handler',
    functionName: name, environment: env,
    onSuccess: new destinations.EventBridgeDestination(eventBus)
  });
}
```

“Serverless automation isn’t about provisioning but about composition and configuration.”

Your Cloud Architect

CDK: Domain modeling for serverless automation

"The AWS Construct Library includes higher-level constructs, which we call *patterns*. These constructs are designed to help you complete common tasks in AWS, often involving multiple kinds of resources."

https://docs.aws.amazon.com/cdk/latest/guide/constructs.html

Business Domain Constructs

- Bank
- Loan Broker
- LoanQuote

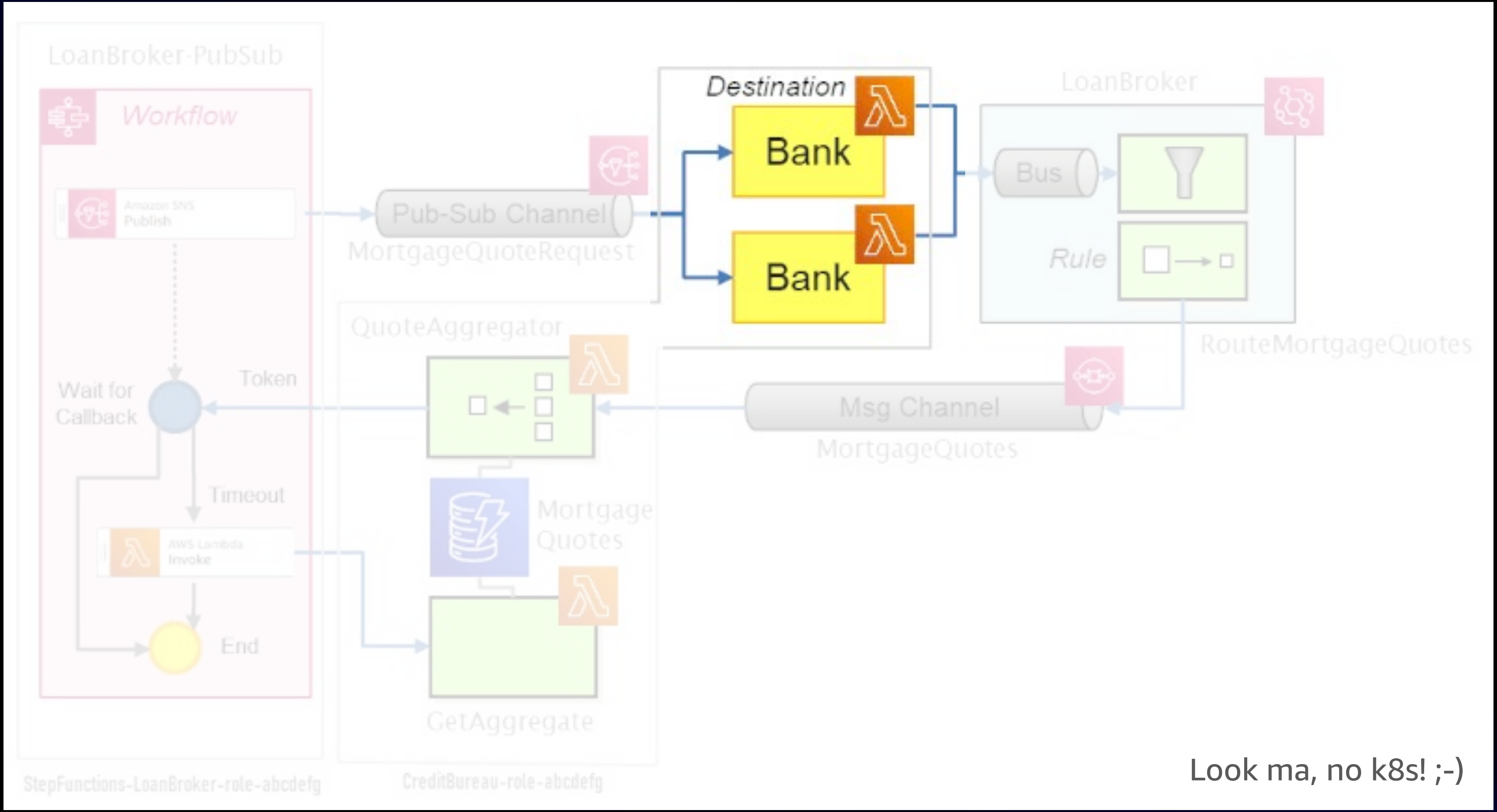
Integration Patterns

- Message Filter
- Content Filter
- Aggregator
- Publish-Subscribe

CDK Constructs (CloudFormation Resources)

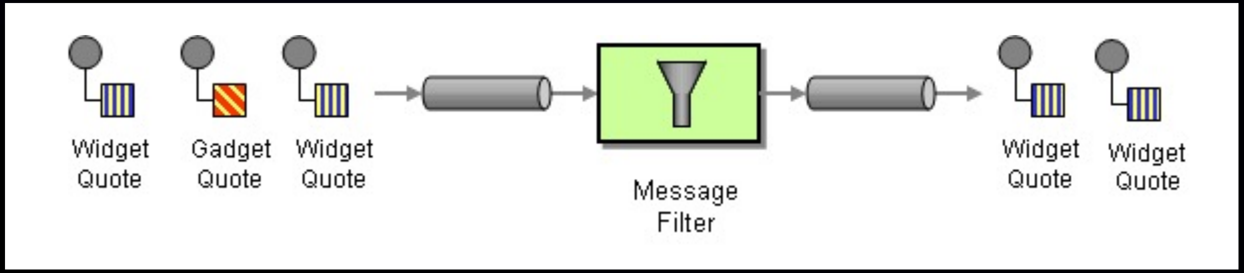
- Lambda Function
- Lambda Destination
- SQS Queues
- Step Function Tasks
- EventBridge Rules

Adding a Message Filter and Content Filter

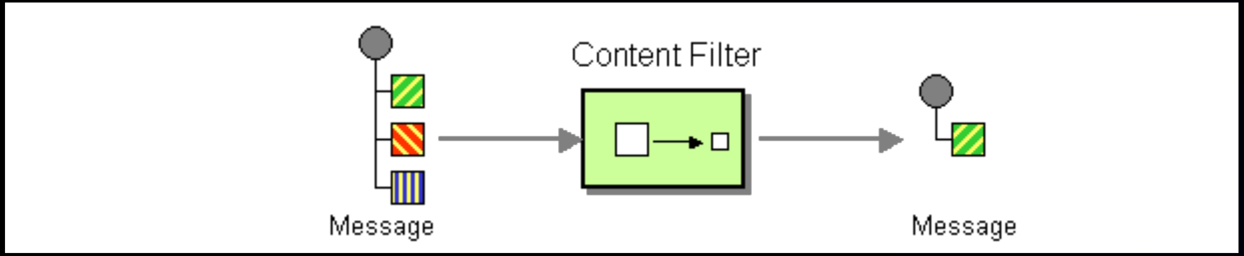


Look ma, no k8s! ;-)

Message Filter and Content Filter



Use a special kind of Message Router, a *Message Filter*, to eliminate undesired messages from a channel based on a set of criteria.



Use a *Content Filter* to remove unimportant data items from a message leaving only important items.

Encoding Integration Patterns with CDK

```
nonEmptyQuotesOnly = MessageFilter.fromDetail(this, 'nonEmptyQuotes’,
    { "responsePayload": {"bankId": [{ "exists": true }] } } );
payloadOnly = ContentFilter.payloadFilter(this, 'PayloadContentFilter’);

new MessageContentFilter(this, 'FilterMortgageQuotes’,
    { sourceEventBus: mortgageQuotesEventBus, targetQueue: mortgageQuotesQueue,
      messageFilter: nonEmptyQuotesOnly, contentFilter: payloadOnly });
```

Message Filter

```
nonEmptyQuotesOnly = MessageFilter.fromDetail(this, 'nonEmptyQuotes’,
    { "responsePayload": {"bankId": [{ "exists": true }] } } );
```

```
interface MessageFilterProps extends EventPattern{}

class MessageFilter extends cdk.Construct {
  public readonly eventPattern: EventPattern;

  constructor(scope: cdk.Construct, id: string, props: MessageFilterProps) {
    super(scope, id); this.eventPattern = props;
  }

  static fromDetail(scope: cdk.Construct, id: string, detailProps: any) : MessageFilter {
    return new MessageFilter(scope, id, { detail: detailProps });
  }
}
```

Content Filter

```
nonEmptyQuotesOnly = MessageFilter.fromDetail(this, 'nonEmptyQuotes’,
    { "responsePayload": {"bankId": [{ "exists": true }] } } );
payloadOnly = ContentFilter.payloadFilter(this, 'PayloadContentFilter’);
```

```
interface ContentFilterProps { readonly jsonPath: string;}

class ContentFilter extends cdk.Construct {
  public readonly ruleTargetInput: RuleTargetInput;

  constructor(scope: cdk.Construct, id: string, props: ContentFilterProps) {
    super(scope, id);
    this.ruleTargetInput = RuleTargetInput.fromEventPath(props.jsonPath)
  }
  static payloadFilter(scope: cdk.Construct, id: string) : ContentFilter {
    return new ContentFilter(scope, id, { jsonPath: '$.detail.responsePayload' });
  }
}
```

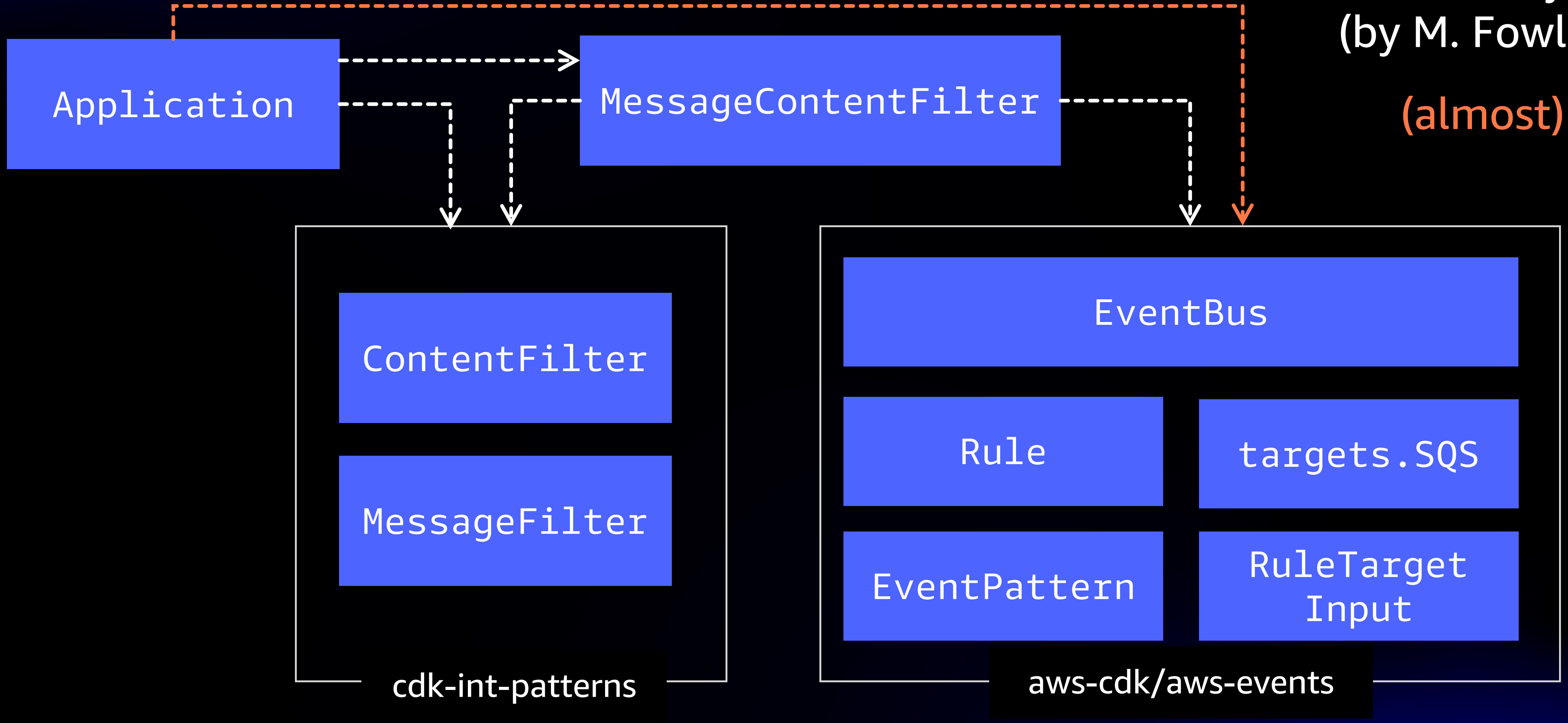
Message & Content Filter → EventBridge

```
nonEmptyQuotesOnly = MessageFilter.fromDetail(this, 'nonEmptyQuotes’,
    { "responsePayload": {"bankId": [{ "exists": true }] } } );
payloadOnly = ContentFilter.payloadFilter(this, 'PayloadContentFilter’);

new MessageContentFilter(this, 'FilterMortgageQuotes’,
    { sourceEventBus: mortgageQuotesEventBus, targetQueue: mortgageQuotesQueue,
      messageFilter: nonEmptyQuotesOnly, contentFilter: payloadOnly });
```

```
class MessageContentFilter extends cdk.Construct {
  public readonly eventPattern: EventPattern;
  constructor(scope: cdk.Construct, id: string, props: MessageContentFilterProps) {
    super(scope, id);
    const rule = new Rule(scope, id + 'Rule’,
      { eventBus: props.sourceEventBus, ruleName: id + 'Rule’ });
    rule.addEventPattern(props.messageFilter.eventPattern);
    rule.addTarget(new targets.SqsQueue(props.targetQueue,
                   {message: prop.contentFilter.ruleTargetInput}));
  }
}
```

Object Structure and Dependencies

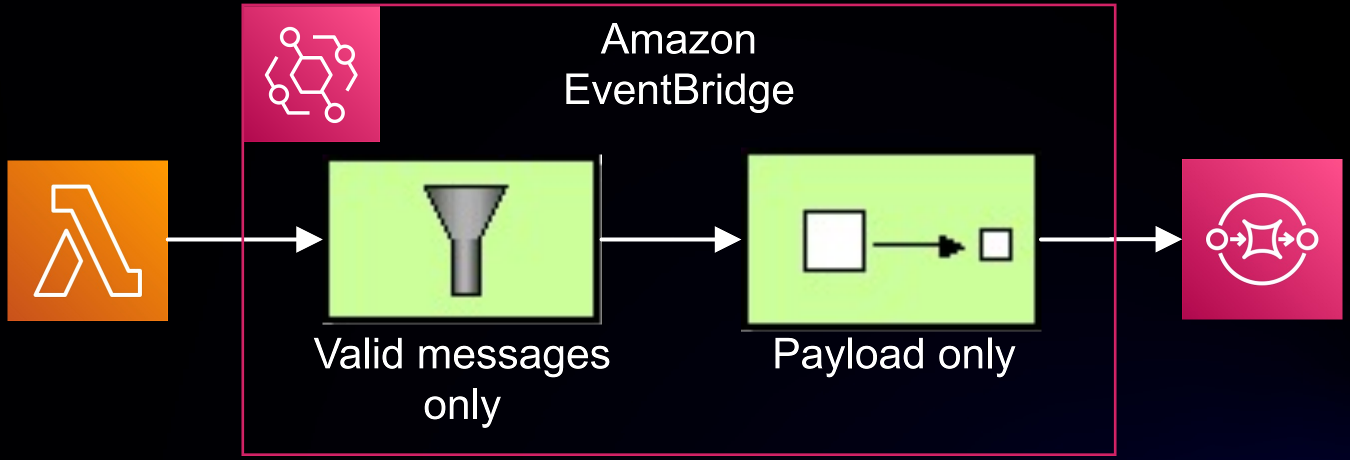


"Gateway"
(by M. Fowler)

(almost)

Taking it a step further

```
new Pipe()
    .attachTo(lambda)
    .append(new MessageFilter('{"bankId": [{ "exists": true }] }'))
    .append(ContentFilter.payloadFilter())
    .publishTo(mortgageQuoteQueue)
    .generateRuntime(scope);
```



Whoa! Isn’t that something?

- Are we deploying, configuring, or programming? All of it!
- We defined a domain-specific language (DSL) for loosely coupled, distributed solutions – that’s what modern cloud apps are!
- We mapped this DSL to AWS CDK and thus make it an executable language to deploy runtime components.
- So we are coding serverless solutions in a domain language!
- No way we could have done this without cloud, serverless, automation, and integration!

"Automation isn't an afterthought. Done right, it impacts your architecture choices and blurs the lines between building and deploying."

Your Modern Cloud Architect

We’re not quite done. Some good questions:

- Can we clean up package dependencies without complicating the code?
- Mapping from patterns to run-time constructs isn’t 1:1. Can we make a smart deploy that places as many patterns into one runtime construct?
- Can we make a mapper that chooses different runtime products for different pattern complexities, e.g. map a simple JsonPath filter to EventBridge but a more fancy one to a Lambda function or StepFunctions?
- Can we regenerate the domain structure from the runtime via tagging and Control Bus events?

+

+

AWS Cloud
Development Kit (AWS CDK)

=

Modern Cloud Application Bliss

Want to learn more?

Welcome to **Serverless** Land

This site brings together all the latest blogs, videos, and training for AWS Serverless. Learn to use and build apps that scale automatically on low-cost, fully-managed serverless architecture.

https://serverlessland.com/reinvent2021/api308 @ghohpe
https://ArchitectElevator.com
https://EnterpriseIntegrationPatterns.com

Thank you!

Gregor Hohpe

@ghohpe
ArchitectElevator.com
EnterpriseIntegrationPatterns.com